Final Project

Thomas Nocera

12/13/2021

Introduction

This project looks at two algorithms that randomizes Thompson Sampling to see how it performs in the worst case in a Bernoulli bandit setting in an attempt to analyze how it would perform in an adversarial setting. The motivation behind the algorithms come from the Exp3 algorithm. Exp3 is analogous to a randomized greedy algorithm because it prioritizes the arm that has given the best rewards so far by creating a probability distribution giving larger probabilities to arms that perform well. Thompson Sampling is an improvement on a greedy algorithm in the stochastic setting since it samples a probable average from a distribution constructed from the observed rewards, rather than just choosing the arm with the best average reward so far. So we investigate randomizing Thompson Sampling so that we get randomization of probable averages, which would explore arms that we are more uncertain about, based on the variance of their modeled distributions, more often than Exp3.

The beta distribution is used in Thompson Sampling to model the distributions of each arm which adds an inherent randomized element to the algorithm in the first place. This randomization is not sufficient in the adversarial case. The variance of the beta distribution for a random variable X and parameters α and β is

$$\text{var}(X) = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$$

which for large enough values of α and β is a decreasing function. Since Thompson Sampling strictly increases both α and β this causes the distributions to continuously tighten. At some point the distributions for each arm will be so tight that Thompson Sampling will begin choosing only one arm. The randomization element of the proposed algorithm attempts to fix this issue.

The environment we are working in is a Bernoulli bandit environment with three arms. We test the algorithm against Thompson Sampling, Greedy, and Exp3 to see how it compares to more established algorithms. Since Exp3 and the proposed algorithm are randomized, we should expect them to do worse in this environment since they will waste pulls on exploring arms that are not optimal. Because of this, we will also analyze the performance on the worst 10% of simulations over a total of 1000 simulations. This is a better model of an adversarial setting since an adversary ideally picks an environment where the algorithm performs its worst.

Proposed Algorithm

For our algorithm we let the time horizon be T, the number of arms be K, and for each $k \in K$ the initial state of α_k and β_k, the parameters for the beta distribution, at $t = 1$ is $\alpha_k = \beta_k = 1$ so that each arm's distribution starts uniformly. We then sample a realistic average reward from $\text{beta}(\alpha_k, \beta_k)$, for each arm k, which we call $\hat{\theta}_k$. At this point the algorithm has mirrored Thompson Sampling, the randomization happens when we choose the arm. We sample arm $x_t \sim \text{softmax}(\hat{\theta})$, pull the arm, and observe reward r_t. The softmax function takes a k-dimensional vector $\boldsymbol{z}$ and constructs a k-dimensional vector $\boldsymbol{P}$ with each component

$$P_i = \frac{e^{\eta z_i}}{\sum_{j=1}^{k} e^{\eta z_j}}$$

This vector $\boldsymbol{P}$ has the property that each component $P_i \in [0, 1]$ and $\sum_{i=1}^{k} P_i = 1$, so that it may be sampled from.

The first proposed algorithm is the most natural way to randomize Thompson Sampling. The only difference being that the chosen arm x_t is being taken from the softmax of the sampled $\hat{\theta}_k$ rather than the argmax. This allows the learner to randomize the arms it chooses while also picking arms that perform well more often. In fact, one could argue this is exactly the same as Thompson Sampling except we have simply changed the distributions we choose from.

Algorithm 1 Probabilistic Thompson Sampling

for $t = 1, \ldots, n$ **do**
  for $k = 1, \ldots, K$ **do**
    Sample $\hat{\theta}_k \sim \text{beta}(\alpha_k, \beta_k)$
  end for
  $\hat{\theta} = [\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_k]^T$
  $x_t \leftarrow \text{softmax}\hat{\theta}$ ▷ This is where we randomize
  Apply x_t and observe r_t
  $\text{beta}(\alpha_{x_t}, \beta_{x_t}) \leftarrow \text{beta}(\alpha_{x_t} + r_t, \beta_{x_t} + 1 - r_t)$
end for

An alternative algorithm involves saving all of the previous samples of $\hat{\theta}_{k,t}$ then we let the components of $\hat{\theta}$ be

$$\hat{\theta}_i = \sum_{j=1}^{t} \hat{\theta}_{i,j}$$

then sample from the softmax of this new vector.

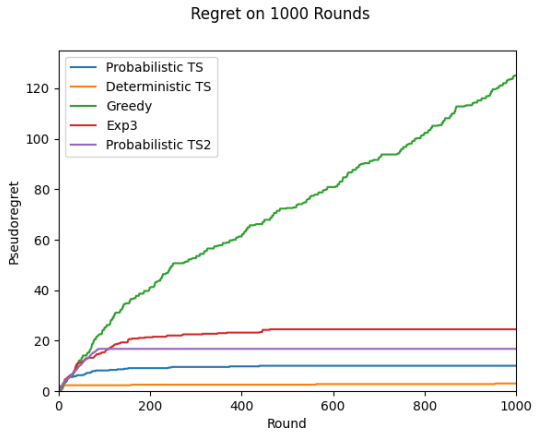
Algorithm 2 Probabilistic Thompson Sampling 2

for $t = 1, \ldots, n$ **do**
  for $k = 1, \ldots, K$ **do**
    Sample $\hat{\theta}_{k,t} \sim \text{beta}(\alpha_k, \beta_k)$
  end for
  $\hat{\theta} = [\sum_{i=1}^{t} \hat{\theta}_{1,i}, \sum_{i=1}^{t} \hat{\theta}_{2,i}, \ldots, \sum_{i=1}^{t} \hat{\theta}_{k,i}]^T$
  $x_t \leftarrow \text{softmax}\hat{\theta}$
  Apply x_t and observe r_t
  $\text{beta}(\alpha_{x_t}, \beta_{x_t}) \leftarrow \text{beta}(\alpha_{x_t} + r_t, \beta_{x_t} + 1 - r_t)$
end for

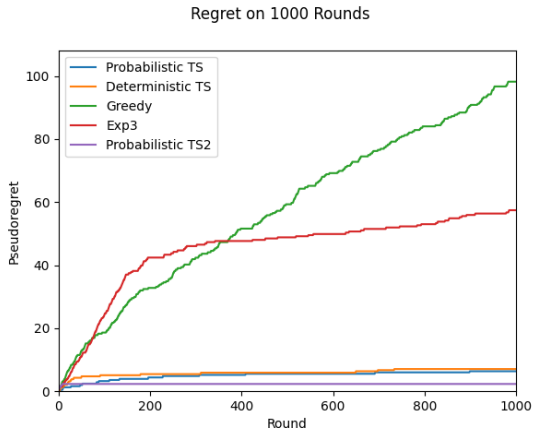
This more closely follows the methods of Exp3. In Exp3, even if an arm is not pulled, the variable used as the estimator for the reward of that an arm is updated. The same thing happens in this algorithm but it continues to update with realistic averages for each arm, rather than with just 1 as Exp3 does. It is important to note that in this algorithm the beta distribution for arms that are not pulled are not updated, so our estimator $\hat{\theta}_k$ becomes more confident of its estimation even though we have received not new information about its reward. I do not think this is a very good adversarial algorithm as it is stated since it becomes more confident over time which is something an adversary can exploit. One might want to implement a decreasing function for the learning rate η to make the algorithm become more randomized in time.

Regret Analysis of a Stochastic Bandit t=1000

We run the algorithm over 1,000 rounds and compare against Thompson Sampling, Greedy, and Exp3. In figure a the means of each arm are randomly generated. Unsurprisingly Thompson Sampling performs well compared to all of the other algorithms. Surprisingly though we see that the Greedy algorithm performs worse than all of the algorithms designed for an adversarial setting. We should expect it to do better since the adversarial algorithms will waste pulls on sub-optimal arms because they try to randomize their choices. In figure b we fix all of the arm means to be relatively close to see how that might affect the outcome. We see Exp3 performs much worse in this case while all of the Thompson Sampling inspired algorithms perform considerably well. Exp3 seems to have trouble picking the correct arm in the beginning when the means of the arms are close together. We surprisingly see the second proposed algorithm performing much better than all of the arm which shouldn't happen due to the attempts at randomizing the algorithm. I will discuss possible issues with this algorithm in an adversarial setting in the conclusion.

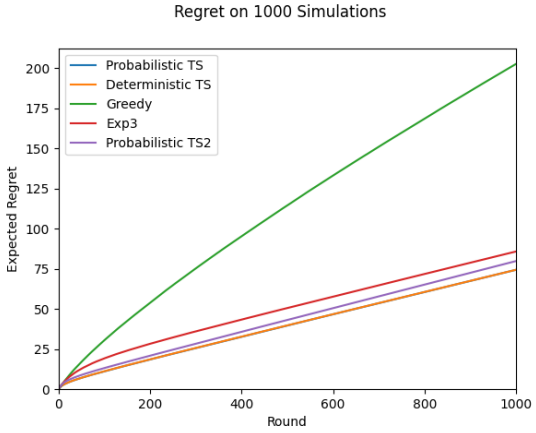


(a) The pseudoregret of four algorithms against a three armed Bernoulli bandit with randomly generated means.

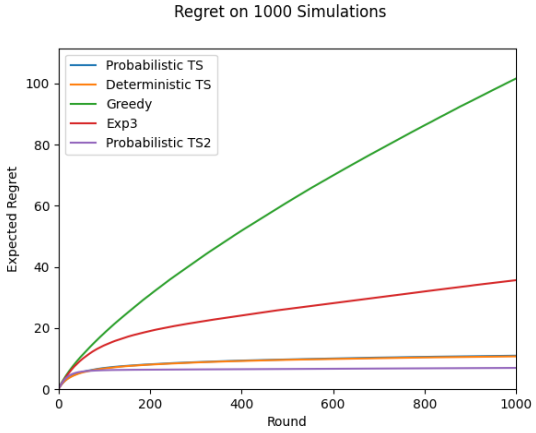


(b) The pseudoregret of four algorithms against a three armed Bernoulli bandit with means 0.5, 0.55, and 0.5 respectively.

Regret Analysis Over 1000 Simulations



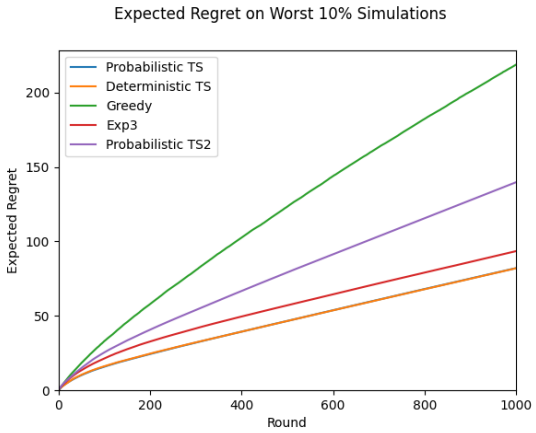
(a) The expected regret of four algorithms against a three armed Bernoulli bandit with randomly generated means ran over 1,000 simulations with $T = 1,000$.



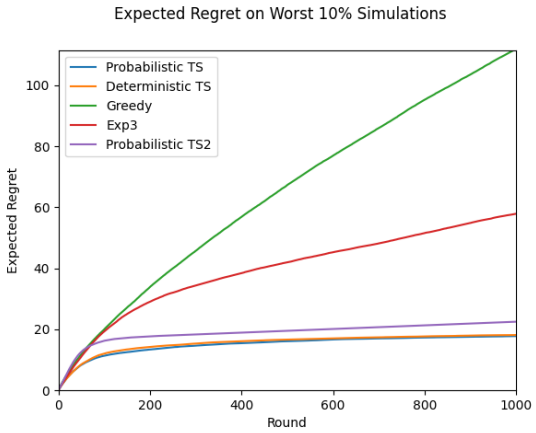
(b) The expected regret of four algorithms against a three armed Bernoulli bandit with means 0.5, 0.55, and 0.5 respectively ran over 1,000 simulations with $T = 1,000$.

We then take a look at how these algorithms perform on average since there is a great deal of variation involved in each run. Each algorithm is run over 1,000 times and the regret at each t is averaged to get the expected regret. In figure a we have generated random means for each of the three arms at every new simulation. Again we see the Greedy algorithm performing the worst and Thompson Sampling performing the best. The first proposed algorithm is hidden under Thompson Sampling following it very closely while the second proposed algorithm performs well but its trajectory looks linear which is not good for the long run. It looks as though Exp3 would perform better if the time horizon was instead 10,000. In figure b, on the other hand, the second proposed algorithm performs very well, outperforming all of the algorithms by a decent margin. I would be interested to know why this algorithm performs so well against arms with means close together as opposed to randomized since intuitively one may think an algorithm would generally perform better when the means have larger gaps, which we can conclude happens more often in a randomized mean simulation.

Regret Analysis of the Worst 10% Simulations



(a) The expected regret of the worst 10% of 1,000 simulations of four algorithms against a three armed Bernoulli bandit with randomly generated means with $T = 1,000$.



(b) The expected regret of the worst 10% of 1,000 simulations of four algorithms against a three armed Bernoulli bandit with means 0.5, 0.55, and 0.5 respectively with $T = 1,000$.

Since it is difficult to get an adversarial setting, looking at how well the algorithm performs in the worst case scenarios is the next best thing. An adversary would ideally choose the worst possible environment that the algorithm would perform in, so the worst 10% will better highlight the performance of each algorithm in an adversarial setting. In figure a we can see again that Thompson Sampling and the first proposed algorithm are lined up with each other but the second proposed algorithm does much worse than Exp3. We would expect Exp3 to perform better than in the 1,000 simulations since it should perform better as the environment gets worse as it's designed for an adversarial setting. In figure b we have a surprising change with the second proposed algorithm and the other Thompson Sampling algorithms performing very well. This is probably due to the fact that the means being fixed are not as bad as they could be when they are completely randomized.

Conclusion and Further Ideas

The two proposed algorithms certainly perform fairly well when compared to other algorithms we have studied but each have quite a bit of work to be done before saying anything definitive. Firstly, there is no formal regret analysis so one can not currently say anything definitive about their bounds. The first proposed algorithm acts extremely similar to normal Thompson Sampling. I think this is due to the fact that the softmax probability distribution ends up being too skewed towards what Thompson Sampling would normally choose. A fix for this would be to update all of the non chosen arms in a different way, since as it runs now just reinforces what we already know about the arm when it does not even pull it. For the second proposed algorithm, there are a few adjustments that could be made. I will admit it was implemented without too much thought since it was an idea I had fairly close to the dead line of the project, but I also think the idea has some merit. I again think the updating of the probability distributions for the softmax should be tweaked for arms that are not pulled, similar to what Exp3 does. Another issue that occurs with this algorithm is that it seems to stick to an arm, sometimes sub-optimally, and does not correct itself. This is evident in the fact that it appears to have a linear regret in the 1,000 random simulations and its extraordinarily flat regret in the fixed means setting. This is not a good sign for an algorithm meant for an adversarial setting. Another option to implement is changing the learning rate to be a decreasing function over time so that an adversary will have a harder time exploiting the algorithm as time goes on. One could even somehow increase the learning rate when the rewards are good and decrease it when the rewards are bad, but then that begs the question what rewards are good or bad? In any case, these algorithms clearly

have some sort of merit to them and they perform well in both the Bernoulli stochastic case and in their worst cases. With some tweaking these algorithms could have some merit in the adversarial setting as well.